Ускорение решателей СЛАУ с помощью графических процессоров

В.Л. Якушев, А.В. Филимонов, П.Ю. Солдатов

Институт автоматизации проектирования РАН, Москва, Россия

yakushev@icad.org.ru, filimonovanton@icad.org.ru, soldatov@icad.org.ru

Аннотация. В данной работе представлен способ адаптации прямого решателя СЛАУ для вычислительных систем, использующих графические ускорители (GPU). Благодаря перенаправлению наиболее ресурсоемких операций на GPU удалось значительно сократить время работы решателя, при этом серьезных изменений кода и алгоритмов не потребовалось. Описан опыт пошагового повышения быстродействия, перечислены проблемы, возникшие при работе с графическими процессорами, и рассмотрены варианты их решения. Также было проанализировано влияние различных факторов на эффективность решателя. Приведены результаты тестирования и намечены направления дальнейшей работы.

Ключевые слова

Графические процессоры (GPU), параллельные вычисления, решение разреженных СЛАУ, разложение Холецкого, системы автоматизированного проектирования.

1 Введение

Благодаря параллельным алгоритмам и развитию архитектур центральных процессоров можно существенно оптимизировать работу решателей. Тем не менее, повышать производительность вычислений на традиционных архитектурах становится все сложнее. Альтернативой являются вычисления на графических процессорах (GPU). Благодаря большому количеству ядер и другим особенностям архитектуры использование GPU позволяет значительно увеличить производительность вычислений для некоторых задач [1]. В данный момент развитие технологий вычислений общего назначения на GPU идет быстрыми темпами, причем как на аппаратном, так и на программном уровне.

В данной работе рассмотрен прямой решатель симметричных разреженных СЛАУ реализует разложение Холецкого. Данный метод является прямым, а значит время работы решателя практически не зависит от количества правых частей. Данная особенность актуальна для применения метода в программных комплексах для расчета сооружений, так как в строительной механике часто возникают задачи с большим количеством правых частей [2, 3, 4]. Решатель эффективно распараллелен для машин с общей памятью с помощью директив OpenMP.

Анализ работы алгоритма показал, что операция умножения матриц занимает 80 —85% от времени факторизации. Используя GPU для умножения матриц, возможно серьезно уменьшить время выполнения умножений и, следовательно, серьезно уменьшить время работы решателя [5].

Решатель использует интерфейс BLAS (Basic Linear Algebra Subprograms), который является де-факто стандартом интерфейса программирования приложений для создания библиотек, выполняющих основные операции линейной алгебры, в том числе умножение матриц (GEMM). В то же время существует библиотека cuBLAS, которая реализует тот же интерфейс, адаптирована для GPU и входит в стандартный комплект разработки CUDA Toolkit, поставляемый компанией nVidia [6, 7]. Таким образом, можно динамически подключить две библиотеки, оптимизированные для разных архитектур. Для внедрения cuBLAS в работу решателя потребовалось разработать набор функций, которые устраняют отличия в обращении к BLAS и cuBLAS (например, в передаваемых типах данных) и корректно осуществляет передачу данных между вычислительными мощностями [8].

2 Поиск оптимального способа умножения матриц на GPU

Из-за особенностей работы GPU применение данного подхода вызывает некоторые сложности. Так, перенаправление абсолютно всех операций умножения матриц не ускорит, а замедлит работу решателя, поскольку будет затрачено слишком много времени на переписывание данных. Поэтому необходимо было определить критерии направления операции на GPU. Отдельный интерес представляет поиск максимально быстрого способа копирования данных на GPU и обратно. Подготовка данных для GPU происходит во многих параллельных потоках на CPU, и при их передаче образуется очередь, которая практически сводит на нет распараллеливание OpenMP. Оптимальный баланс, полученный на определенном примере и конфигурации оборудования, может быть нарушен при изменении конфигурации или задачи.

В ходе работы были разработаны тестовые программы сравнения скорости умножения матриц. По результатам их выполнения были определены размеры массивов, превышение которых могло сделать выгодным использование GPU. Очевидно, чем выше класс используемого GPU, тем ниже находится этот порог, тем быстрее выполняются сами вычисления, и уменьшается время работы решателя. Даже с учетом копирования данных на GPU и обратно для матриц, содержащих десятки тысяч элементов, умножение матриц на GPU может выполняться на один-два порядка быстрее, чем при использовании многопоточных CPU, устанавливаемых в настольных ПК.

2.1 Копирование данных

Любое обращение к GPU требует достаточно много времени. Так, было установлено, что выделять и освобождать участок памяти на GPU для каждого набора переменных неэффективно. Поэтому при инициализации устройства под указатели выделяется память, размер которой сопоставим с доступной глобальной памятью GPU, копирование матриц производится каждый раз в один и тот же участок памяти, а освобождение памяти производится по завершении работы устройства. Также установлено, что при необходимости выполнения операций с подматрицами передаваемых матриц выгоднее переписать матрицу полностью, а не только нужную для вычислений часть, т.е. многократный вызов копирования, пусть даже и незначительных объемов данных, серьезно сказывается на времени работы.

В многопоточном режиме образование очередей заданий для GPU замедляет работу решателя, поскольку стандартными способами отправить новое задание на GPU можно только после завершения исполнения текущего. Быстрое умножение не перекрывает времени ожидания и копирования, и уменьшить время работы решателя не удается. В случае, если в алгоритме распределения прописать запрет на формирование заданий для GPU, когда GPU уже занят, то эффект от работы GPU будет нивелирован относительно медленной работой ядер CPU с оставшимися в их распоряжении большими объемами данных. Тем не менее, удается достичь незначительного ускорения: около 20% в четырехпоточном режиме и около 50% в однопоточном.

Запуск профилировщика GPU для тестового примера умножения матриц, аналог которого был встроен в решатель, показал, что загрузить видеокарту должным образом не получается, так как слишком мало ядер участвует в вычислениях, и соотношение времени вычислений к общему времени работы GPU незначительно (рис. 1). Несмотря на экономию времени за счет выделения и освобождения памяти, копирование выполнялось слишком долго. Поэтому был сделан вывод о неэффективности библиотеки, формирующей и передающей задания для GPU, и необходимости ее усовершенствования. Для получения приемлемого результата умножение матриц на GPU должно выполняться гораздо быстрее.

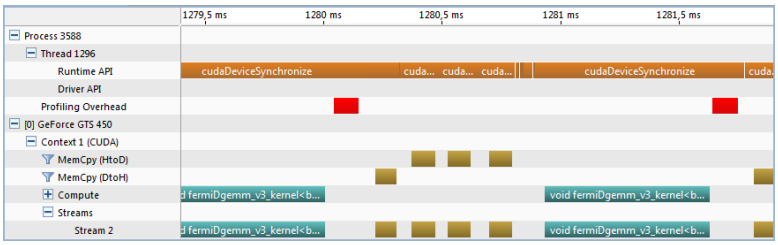

Рис. 1. Профилирование умножения матриц на GPU без оптимизации.

С целью улучшения взаимодействия между CPU и GPU была применена pinned-память — прикрепленный буфер в оперативной памяти, который очень быстро может быть помещен в память GPU. Каждая

матрица, которую нужно записать на GPU, сначала помещается в pinned-массив, а затем асинхронно копируется на GPU. Также было применено асинхронное копирование. Тестовый пример показал, что видеокарта стала простаивать гораздо реже (рис. 2), а скорость выполнения умножения, включая копирование матриц, значительно улучшилась по сравнению с реализацией cublasDgemm без применения оптимизации (рис. 3).

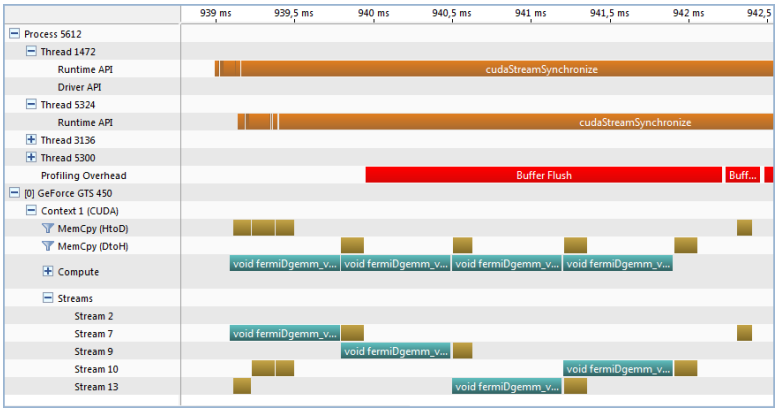

Рис. 2. Профилирование умножения матриц на GPU без оптимизации.

Рис. 3. Сравнение разных способов умножения матриц.

2.2 Cuda streams

Работа решателя, и в том числе подготовка заданий для GPU, происходит во многих параллельных потоках CPU. Таким образом, актуально использовать способы распараллеливания на самом графическом процессоре [9]. Благодаря cuda streams удается выполнять на видеокарте несколько операций умножения практически одновременно: каждый omp thread создает свой поток на GPU, и возникает параллельность выполнения некоторых команд.

Так как программа после отправки задания на GPU продолжает выполнение команд на CPU, количество одновременно работающих потоков на GPU может превышать количество потоков omp, что может привести к переполнению памяти GPU и замедлению работы. Чтобы избежать этого, требуется ограничивать количество cuda streams, и направлять лишние задания на CPU. Оптимальное число потоков на видеокарте также сильно зависит от типа карты.

3 Результаты тестирования

Для настройки алгоритма, проверки правильности и оценки эффективности его работы был подобран ряд конечно-элементных моделей проектируемых строительных объектов из практики ЦНИИСК имени В.А. Кучеренко (рис. 4).

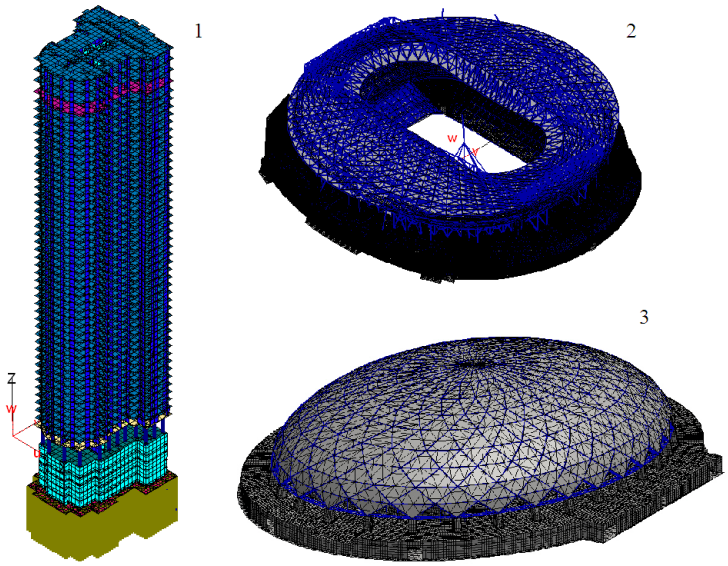

Рис. 4. Примеры моделей, использовавшихся при тестировании.

Численные эксперименты были проведены для различных конфигураций оборудования. Результаты тестирования решения — время расчета для каждой модели в различных режимах работы — приведены в табл. 1. Все данные получены для двойной точности вычислений. Тестирование данного решателя производилось на оборудовании низкого и среднего ценового диапазона.

Таблица 1. Результаты тестирования (конфигурация 1: CPU Intel Core i7 3400 MHz (4 ядра), GPU nVidia GeForce GTX 550Ti; конфигурация 2: 2x Intel Xeon X5680 3300 MHz (6 ядер), GPU nVidia Tesla M2090)

		Время выполнения факторизации, с					
		Конфигурация 1			Конфигурация 2		
Номер задачи	Количество степеней свободы	CPU	CPU+GPU	Ускорение	CPU	CPU+GPU	Ускорение
1	4 870 800	53	35	1.51	43	28	1.53
2	889 890	29	19	1.52	28	16	1.75
3	2 534 446	44	30	1.47	40	22	1.81
4	397 950	127	85	1.49	120	48	2.5
5	2 545 314	267	127	2.1	141	53	2.66

4 Заключение

При использовании графических ускорителей удалось уменьшить время работы решателя на 35 —60%. Дальнейшее развитие представленного подхода видится в использовании нескольких графических процессоров. Рассматривается вариант написания собственного ядра для перемножения небольших матриц, так как время работы функций cuBLAS на небольших объемах данных непредсказуемо. Возможно варьировать способы копирования в зависимости от размеров входящих матриц (в частности, проводить запись нескольких передаваемых матриц в одну). Также необходимо стремиться к максимальному использованию

пропускной способности канала передачи данных между вычислительными мощностями. Из-за сильной зависимости производительности от класса используемого оборудования требуется уделить особое внимание алгоритму автоматической настройки под конкретную конфигурацию. Решатель планируется к использованию в расчетных программных комплексах.

Список литературы

[1] Cullinan C., Wyant C., Frattesi T. Computing Performance Benchmarks among CPU, GPU, and FPGA. Available at http://www.wpi.edu/.

[2] Якушев В.Л., Жук Ю.Н., Симбиркин В.Н., Филимонов А.В. Реализация методов расчета для большеразмерных задач строительной механики в программном комплексе STARK ES. *Вестник кибернетики 2011.* № 10. С. 109–116.

[3] Якушев В.Л., Симбиркин В.Н., Филимонов А.В. Решение большеразмерных задач строительной механики методом конечных элементов в программном комплексе STARK ES. *Теория и практика расчета зданий, сооружений и элементов конструкций. Аналитические и численные методы: Сб. трудов международной научно-практической конференции.* Москва: МГСУ, 2010. С. 516–526.

[4] Yakushev V.L., Shuk U.N., Simbirkin V.N., Novikov P.A., Filimonov A.V. Solution of the generalized eigenvalue problem for 3D tensile structures in Stark ES. *TensiNet symposium 2010 Tensile Architecture: Connecting Past and Future* held 16-17-18 September 2010 at Sofia, University of Architecture, Civil Engineering and Geodesy. P. 103-104.

[5] Tan G., Li L., Triechle S., Phillips E., Bao Y., Sun N. Fast implementation of DGEMM on Fermi GPU. *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, ACM*, New York, NY, USA , pp. 35:1–35:11.

[6] Sanders J., Kandrot E. CUDA by Example: an Introduction to General Purpose GPU Programming. // *NVIDIA Corporation*. Available at http://developer.nvidia.com/

[7] CUBLAS Library User Guide. Available at http://developer.nvidia.com/

[8] Якушев В.Л., Филимонов А.В., Новиков П.А., Солдатов П.Ю. Создание эффективных решателей для GPU. *Научн.-практ. конф. с междунар. участием с элементами научн. шк. для молодежи Высокопроизводительные вычисления на графических процессорах:* Тез. докл., Пермь, 2012 г. С. 85–87.

[9] CUDA C Programming Guide. Available at http://docs.nvidia.com/